Army Research Laboratory

Guidance, Navigation, and Control System Simulations via Graphics Processor Unit

by Mark Ilg

ARL-TR-5645 September 2011

Army Research Laboratory
Aberdeen Proving Ground, MD 21005

ARL-TR-5645 **September 2011**

Guidance, Navigation, and Control System Simulations via Graphics Processor Unit

Mark Ilg
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* September 2011	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** January to June 2011

4. TITLE AND SUBTITLE Guidance, Navigation, and Control System Simulations via Graphics Processor Unit	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Mark Ilg	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-WML-F Aberdeen Proving Ground, MD 21005	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-5645
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Monte Carlo simulation is crucial in the design, development, and execution of a guided projectile program. Graphics processing units (GPUs) are powerful parallel computing devices that are increasingly being used for general purpose (GP) computing. This technical report details the use of GPUs for Monte Carlo simulations with the goal of aiding the Guidance, Navigation, and Control (GN&C) engineer during the design phase of a guided weapon. This report provides a brief overview of GP GPU computing, a basic six-degree-of-freedom projectile dynamic model, and the implementation of a GPU. Run-time performance comparisons are performed between serial Monte Carlo simulations performed on a central processing unit (CPU) and parallel simulations performed on a GPU. The results show that for large numbers of trajectories, significant run-time reductions are possible for Monte Carlo simulations performed on the GPU in comparison to simulations performed serially on the CPU.

15. SUBJECT TERMS

GN&C, GPU, Monte-Carlo

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Mark Ilg
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	18	**19b. TELEPHONE NUMBER** *(Include area code)* (410) 306-0780

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction

Our traditional modeling and simulation (M&S) tools can no longer sustain the pace of technological advances and required a new and truly multidiscipline approach, integrating expertise across a variety of domains. Model simplification and back-of-the-envelope calculations are no longer sufficient for complex systems and designs with constrained time frames and costly experiments. To address these issues, a Model Based Design (MBD) approach is used in order to maintain a work environment that adapts to the ever-changing design.

MBD is a method of designing complex control systems using mathematical and visual tools to address the common problems associated with complex control system designs. MBD is a methodology used in many fields that rely heavily on unique and often demanding control systems that require real-time implementation including the aerospace, automotive, and industrial fields. MBD allows the control engineer to maintain a work environment that adapts to the ever-changing physical aspects of a control system. MBD is significantly different from traditional design processes and begins with a plant model based on the projectile aerodynamics and physical properties consisting of recursive steps and model refinement loops. After model validated through experimentation, computational fluid dynamics, and engineering intuition, a controller is designed to meet the system requirements. The controller is ported from the simulation environment to a processor and the system undergoes processor-in-the-loop (PIL) testing followed by hardware-in-the-loop (HIL) testing using rapid prototyped hardware. The testing and verification process is improved because the entire design life cycle is fluid and does not require significant changes mid-cycle. The dynamic effects on the system are rapidly identified through the HIL testing much more efficiently than with tradition design methodology and problems can be identified early before costly flight testing. This methodology was used in the design of a Guidance, Navigation, and Control (GN&C) system for the Very Affordable Precision Projectile (VAPP) demonstration program.

A demand for increased graphics processing performance is ever increasing in the computing world. To satisfy this demand, the graphics processing unit (GPU) market was developed, originally to meet for the heavy computational requirements of texture mapping and rendering. More recently, the GPU has been tasked for more computational complex operations including geometric calculations. Due to the computing requirements, more and more operations that were traditionally performed in the fixed point arena using dedicated circuits have migrated over to floating point cores on the GPU. NVIDIA, one of the largest producers of GPUs, released a parallel computing architecture specifically designed to leverage the parallel floating point architecture on the GPU, aptly named CUDA. CUDA, released in 2006, allowed programmers familiar with standard programming languages to exploit the capabilities of the GPU directly. CUDA consists of a set of application programming interfaces (API) that allow programmers to execute portions of their code in parallel threads on the GPU's hundreds of cores. The API provides hardware abstraction, which allows software to run on any NVIDIA GPU without specific hardware programming calls.

NVIDIA's release of the CUDA architecture has led to a veritable explosion of GPU-based research by scientists, engineers, and mathematicians over the past five years. Investigation of difficult parallel computational problems, until then restricted to those with access to large-scale computing clusters, were suddenly solvable in reasonable amounts of time using personal computers. Code modifications were minimal due to the straightforward API provided by the architecture.

The following report describes one aspect of the MBD GN&C system design, the Monte Carlo statistical analysis. The report describes the use of a GPU for Monte Carlo analysis to provide an understanding of the ballistic dispersion for a projectile due to various parameter variations. Monte Carlo analysis provides the ability to analyze the projectiles behavior and characterize which parameters are most sensitive during the design process.

2. Mathematical Model

The model used in the simulation and Monte Carlo analysis of the projectile dynamics is depicted in the block diagram outlined in figure 1.

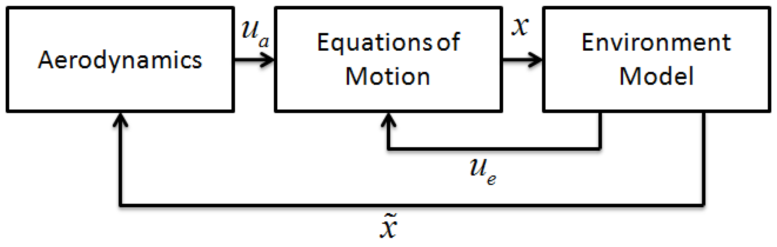

Figure 1. Block diagram of the model.

Note:

u_a	Aerodynamic Forces and Moments
u_e	Environmental Forces and Moments
x	State Vector
$\tilde{x}$	Environmentally Perturbed State Vector

2.1 Equations of Motion

The state equations for the equations of motion block are shown in equations 1–4. Equations 1 and 2 are the Translational and Rotational Dynamic equations for a rigid body, respectively.

$$\dot{V} = \frac{F_b}{m} - \omega \times V \tag{1}$$

$$\dot{\omega} = I^{-1}\left(M_b - \omega \times I\omega\right) \tag{2}$$

where

V	$=$	body fixed velocity vector
F_b	$=$	applied forces
ω	$=$	body fixed angular rates
m	$=$	mass of the projectile
I	$=$	inertia tensor
M_b	$=$	applied moments

The applied forces consist of the body and canard aerodynamic forces and force due to gravity. The applied moments consist of the body and canard moments and the moment due to the center of gravity offset. In the simulation, the aerodynamic forces and moments consist of lookup tables derived through empirical methods, computational fluid dynamics (CFD), wind tunnel experiments, and flight experiments. The remaining state variables consist of the kinematic equations:

$$\dot{X}_e = R^{-1}V \tag{3}$$

$$\dot{\mathbf{q}} = \omega \otimes \mathbf{q} \tag{4}$$

where X_e is the position of the projectile in the reference coordinate system, R is the direction cosine matrix, and $\mathbf{q}$ is the quaternion, and $\otimes$ represents the skew symmetric matrix product. Although quaternion representation is used in this simulation, Euler angles, or direction cosine matrix state propagation could be used. Derivation of the equations of motion has been extensively studied in references 1–6. The aerodynamic model also includes the following sub-models, which will not be detailed: wind models, a gravity model, a temperature model, and a pressure model.

2.2 State Vector Propagation

In order to resolve the motion for a particular instance in time, a ordinary differential equation (ODE) integration method must be used. The state equations of a projectile can be written as a series of nonlinear differential equations as a single point boundry condition initial value problem:

$$\frac{dx_i(t)}{dt} = f_i\left(t, x_o, ..., x_{N-1}\right) \tag{5}$$

where, x_i are the state variables and f_i are known functions of x and t. To propagate these equations, we require a numerical integration technique. Simple methods for initial value problems include Euler Integration, Trapezoidal Integration, and Runge-Kutta. In this work, an adaptive 4th-order Runge-Kutta method is chosen due to ease of coding and computational speed (*7*) . The Runge-Kutta method uses a linear combination of several Euler style integration steps to solve for the state. The formula for the Runge-Kutta method is

$$k_1 = hf(t_n, x_n) \tag{6a}$$

$$k_2 = hf(t_n + \frac{\delta t}{2}, x_n + \frac{k_1}{2}) \tag{6b}$$

$$k_3 = hf(t_n + \frac{\delta t}{2}, x_n + \frac{k_2}{2}) \tag{6c}$$

$$k_4 = hf(t_n + \delta t, x_n + k_3) \tag{6d}$$

$$x_{n+1} = x_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \tag{6e}$$

To choose the time step for the solver, δt, we evaluate the Runge-Kutta solution to the dynamics equations a series of times as outlined in the following steps.

Step 1: Evaluate the solution at the current time step.

$$x_{n^+} = f_{rk}\left(t + \frac{\delta t}{2}, x_n\right) \tag{7}$$

Step 2: Evaluate the solution at the half the current time step, followed by the another iteration of half the time step.

$$x^-_{n+\frac{1}{2}} = f_{rk}\left(t + \frac{\delta t}{2}, x_n\right) \tag{8a}$$

$$x^-_n = f_{rk}\left(t + \frac{\delta t}{2}, x^-_{n+1/2}\right) \tag{8b}$$

Step 3: Compare the residuals of the two simulations and compute the maximum error.

$$e_{\max} = \max\left(x^+_n - x^-_n\right) \tag{9}$$

Step 4: Adjust the next time step.

$$\delta^+_t = \delta^-_t \left(\frac{e_{\max}}{c_{\max}}\right)^{\alpha} \tag{10}$$

Constants c_{max} and α are chosen by the designer based on the dynamics and δ_t is chosen with the following constraints, δ_{tmax} and δ_{tmin}.

$$\delta_t = \begin{cases} \delta_{t\max} & \delta^+_t > \delta_{t\max} \\ \delta_{t\min} & \delta^+_t < \delta_{t\min} \\ \delta^+_t & otherwise \end{cases} \tag{11}$$

3. GPU Processing

This "single-program-multiple-data" structure of the dynamics model is ideal for implementation on a GPU since each simulation is independent of the results of any other simulation. The CPU interacts with the GPU through a memory transfer mechanism within CUDA. Figure 2 shows that overview block diagram of a GPU that uses the CUDA programming API and how it interfaces the CPU. The parallel code, or kernel, is launched and executed on a device by multiple threads. These threads are grouped into blocks, and blocks into grids. Each of the threads consist of a unique simulation model, which is free to execute its own code path (*8*). Although possible, the thread blocks do not communicate via the shared memory or synchronize their execution.

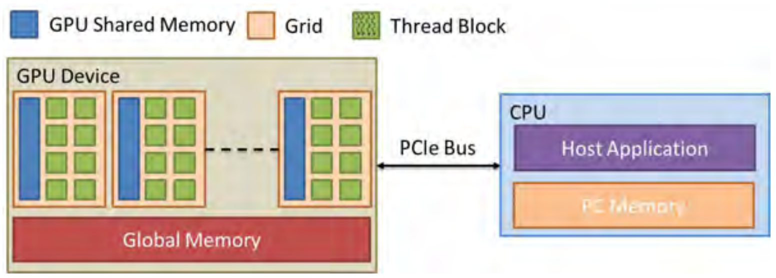

Figure 2. CUDA GPU programming model.

The GPU-based Monte Carlo simulation is described in figure 3. First, the projectile data set is loaded from a file or through another software interface. Next, a set of N initial conditions are generated for the projectile under test. These initial conditions consist of a random number distribution consisting of state vector, mass properties, environmental terms, and aerodynamic coefficient variations. An example of some of the distributions are given in table 1.

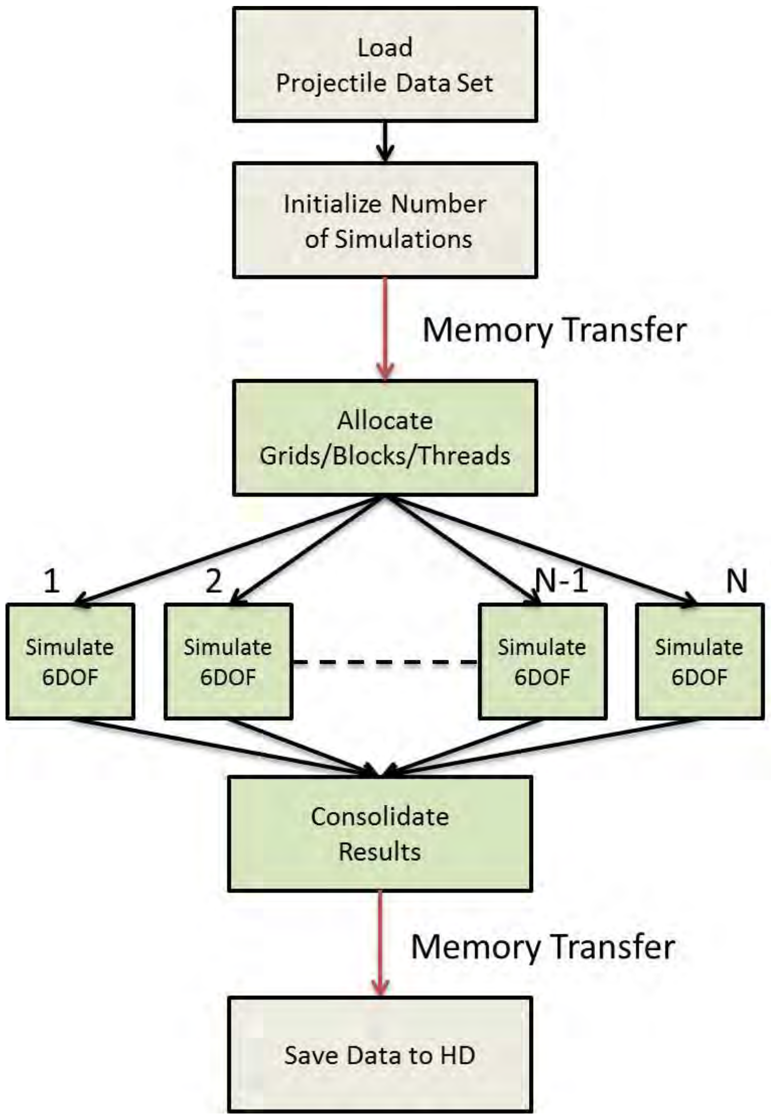

Figure 3. Program flow.

Table 1. Example distributions.

Parameters	Distribution Type	
Aerodynamic Terms	Multiplicative Normal Distribution	$x_N = x * \mathcal{N}(\mu, \sigma)$
Mass Properties	Additive Normal Distribution	$x_N = x + \mathcal{N}(\mu, \sigma)$
Wind	Uniform Distribution	$x_N = \mathcal{U}(min, max)$
Body Initial Conditions	Additive Normal Distribution	$x_N = x + \mathcal{N}(\mu, \sigma)$

These initial conditions are generated using unique probability distribution functions using real-world statistical measurements. The CPU initializes the set of initial conditions, places them in GPU memory, and spawns GPU threads to perform the dynamic simulations as described in section 2. Each GPU thread, executed on a single core of the GPU, simulates the trajectory of one projectile. On completion of the trajectory, flagged by impact or angle of attack error, the central processing unit (CPU) collects and organizes all trajectory data from GPU memory.

4. Results

To show the potential of the GPU as a Monte Carlo tool for guided munitions, comparisons were made between a GPU and a computing cluster (*9*). The six-degrees-of-freedom (6DOF) model of a guided mortar was used in the analysis, developed in the U.S. Army Research Laboratory (ARL) Precision Simulation Environment (PRESIMEN). The parameters for the model, including aerodynamics, wind terms, mass properties, environmental models, etc., were loaded into the GPU data set using the Mathworks API to ensure a fair comparison of the models.

Figure 4 shows the experimental data flow of the computing cluster versus the GPU simulations. The simulation model allows the generic 6DOF model hosted in both the GPU and cluster environments to share similar parameters and initial conditions. To compute the distributions, we used the Mathworks built-in random number generator in the cluster configurations, and for the GPU simulations, the standard math library random number generator was chosen. All double precision values were converted to single point precision prior to passing data from the computed distributions to the GPU. Table 1 provides details about the types of distributions used in the analysis. The distributions were chosen as Normal and Uniform due to the simplicity of implementations and for demonstration purposes.

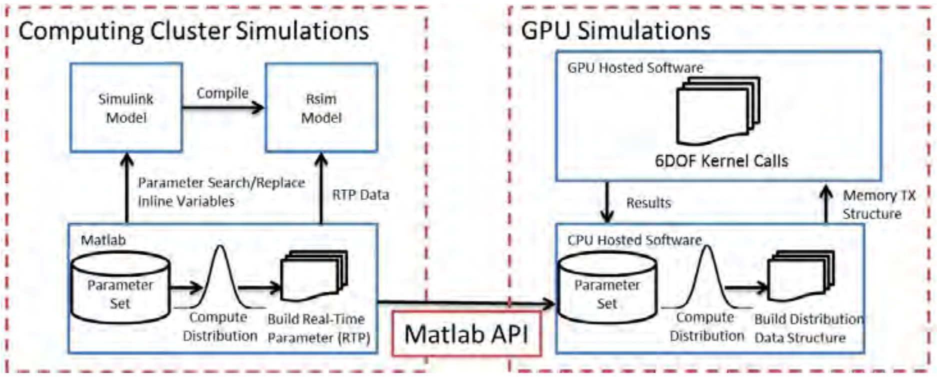

Figure 4. Simulation configuration.

The GPU used in the analysis is a laptop GPU, NVIDIA Quadro FX 2700M, and all software was coded in C++ and CUDA. The GPU is a mid-range mobile workstation graphics card with 48 cores and 512 MB or DDR3 local memory. The GPU grid sized used in the runs is 64 and was determined via experimentation. The workstation is a Dell Precision M6400 laptop with a Intel Core2 Duo Processor at 2.66 GHz, 8 GB DDR3 RAM, and running Windows Vista x64. The computing cluster used in this experiment consists of five Dell Precision T7400 workstations running Vista x64 with four Intel Xeon Processors at 3 GHz with 4 GB RAM each. The computing cluster is connected via 100-MB ethernet and is scheduled using the Condor scheduler using normal priority on all jobs.

Figure 5 shows the histogram of 20k runs on the computing cluster of the 6DOF model. The average execution time of simulation is 13.2 s for a total execution time of 108.53 min.

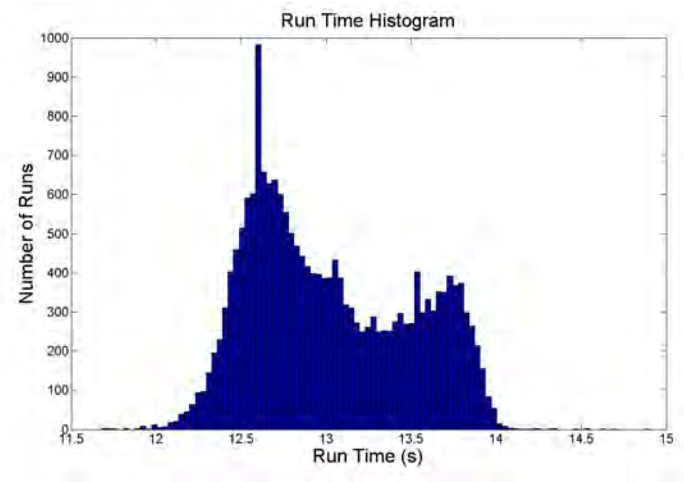

Figure 5. Cluster results.

Figure 6 shows a plot of the number of runs versus the average execution time of the 6DOF, where the number of runs was varied between N=100 and N=100k runs. The average execution time of the 6DOF for approximately 20k runs is approximately 0.13 s, a 100 times improvement over the cluster-based method. The total execution time, including CPU arbitration of the initial distributions, is approximately 12 min. This is a large improvement over the CPU-based method, which inherently has the problem of the ethernet latency. The impact histogram of 100k runs is shown in figure 7. The impact points show a clear Gaussian distribution indicative of a projectile's impact. These data have similar statistical properties to the computing cluster and are useful in system effectiveness studies.

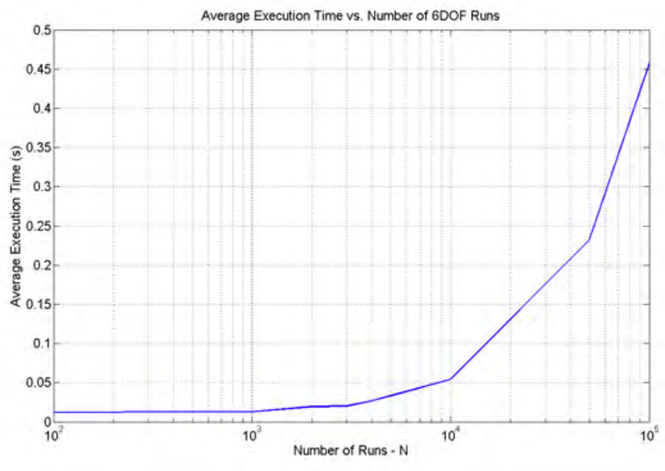

Figure 6. GPU execution time.

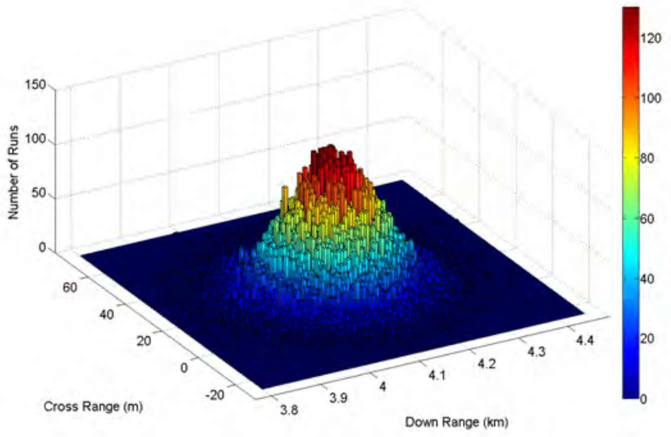

Figure 7. GPU impact distribution.

5. Conclusion

This report proposed a means of using a GPU for simulations of guided projectiles. The proposed method shows that the GPU-based approach has large advantages in computing massive Monte Carlo analysis over a computing cluster using a CPU-only approach. With the short-duration run times of this technology, this method shows enormous prospects in real-time computations of ballistic and guided projectiles for fire-control systems, impact point predictors, particle filters, etc.

References

[1] Ilg, M. Guidance, Navigation, and Control for Munitions, Ph.D. dissertation, Drexel University, 2008.

[2] Wilson, M. Projectile Navigation and the Application to Magnetometers, Ph.D. dissertation, University of Deleware, 2007.

[3] Murphy, C. *Free Flight Motion of Symmetric Missiles*; Brl-r1216; U.S. Ballistics Research Laboratory, July 1963.

[4] Bradley, J. *Equations of Motion - A Prelude to r1216*; U.S. Ballistics Research Laboratory, December 1992.

[5] Hainz III, L.; Costello, M. Modified Projectile Linear Theory for Rapid Trajectory Prediction. *Journal of Guidance, Control, and Dynamics* **2005**, *28* (5), 1006–1014.

[6] Fraysee, J.; Ohlmeyer, E.; Pepitone, T. Guidance, Navigation and Control Without Gyros: A Gun-launched Munition Concept. *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 5–8, 2002.

[7] Press, W.; Teukolsky, S.; Vetterling, W.; Flannery, B. *Numerical Recipes in C++*; Cambridge University Press, 2002.

[8] NVIDIA, Cuda website. *developer.nvidia.com/cuda-training*, 2011.

[9] Ilg, M. *Multi-core Computing Cluster for Safety Fan Analysis of Guided Projectiles*; ARL-TR-5646; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, in press.

List of Symbols, Abbreviations, and Acronyms

6DOF	six-degrees-of-freedom
API	application programming interfaces
ARL	U.S. Army Research Laboratory
CFD	computational fluid dynamics
CPU	central processing unit
GN&C	Guidance, Navigation, and Control
GPU	graphics processing unit
HIL	hardware-in-the-loop
M&S	modeling and simulation
MBD	Model Based Design
ODE	ordinary differential equation
PIL	processor-in-the-loop
PRESIMEN	Precision Simulation Environment
VAPP	Very Affordable Precision Projectile

NO. OF COPIES	ORGANIZATION
1 ELEC	ADMNSTR DEFNS TECHL INFO CTR ATTN DTIC OCP 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1 CD	OFC OF THE SECY OF DEFNS ATTN ODDRE (R&AT) THE PENTAGON WASHINGTON DC 20301-3080
1	US ARMY RSRCH DEV AND ENGRG CMND ARMAMENT RSRCH DEV & ENGRG CTR ARMAMENT ENGRG & TECHNLGY CTR ATTN AMSRD AAR AEF T J MATTS BLDG 305 ABERDEEN PROVING GROUND MD 21005-5001
1	US ARMY INFO SYS ENGRG CMND ATTN AMSEL IE TD A RIVERA FT HUACHUCA AZ 85613-5300
1	COMMANDER US ARMY RDECOM ATTN AMSRD AMR W C MCCORKLE 5400 FOWLER RD REDSTONE ARSENAL AL 35898-5000
1	US GOVERNMENT PRINT OFF DEPOSITORY RECEIVING SECTION ATTN MAIL STOP IDAD J TATE 732 NORTH CAPITOL ST NW WASHINGTON DC 20402
1	US ARMY RSRCH LAB ATTN RDRL WML F M ILG BLDG 4600 ABERDEEN PROVING GROUND MD 21005
3	US ARMY RSRCH LAB ATTN IMNE ALC HRR MAIL & RECORDS MGMT ATTN RDRL CIO LL TECHL LIB ATTN RDRL CIO MT TECHL PUB ADELPHI MD 20783-1197